Obesity – Causes and Consequences

PYTHON FOR DATA ANALYSIS

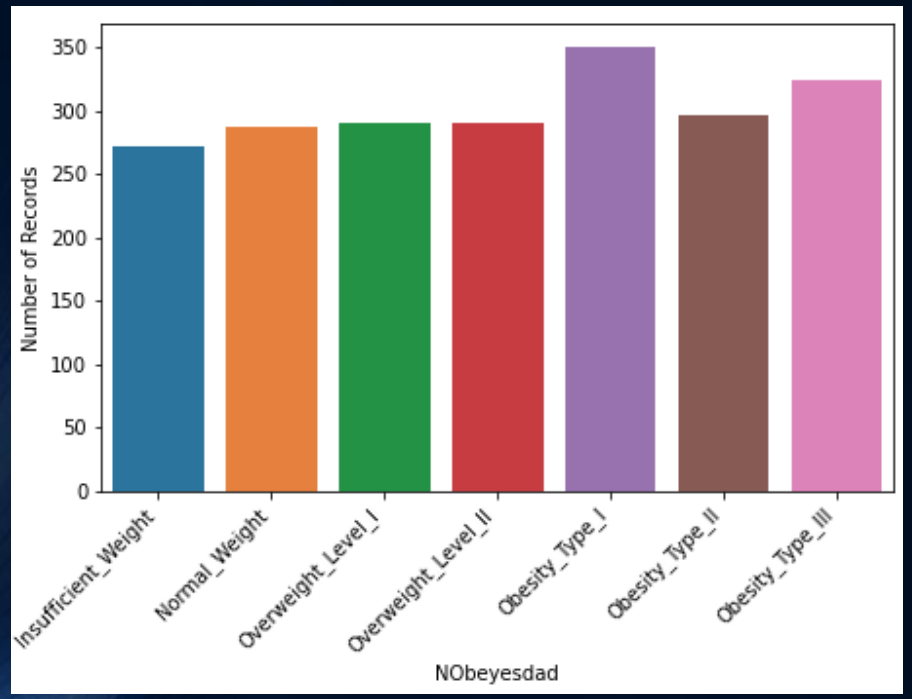
WU SHUTAO – ROUX THOMAS

Objectifs du projet :

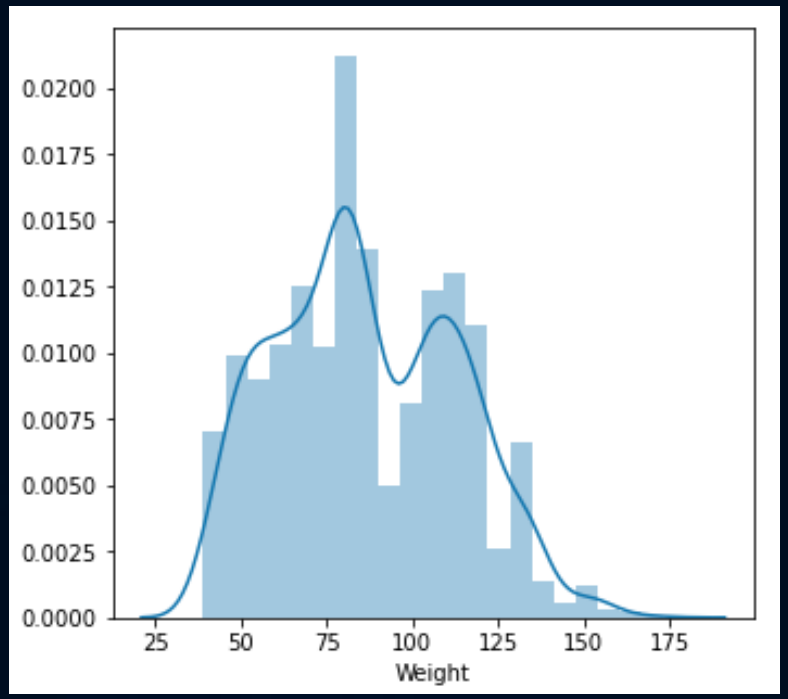
- Phase d'exploration : découverte du DataSet / confrontation variables-cible
- Simplification du DataSet
- Création de modèles
- Confrontation des modèles
- Changement des hyper paramètres
- Conclusion du modèle à retenir
- API Django

I. Exploration du DataSet

NOMBRE DE CAS EN FONCTION DU TYPE D'OBÉSITÉ



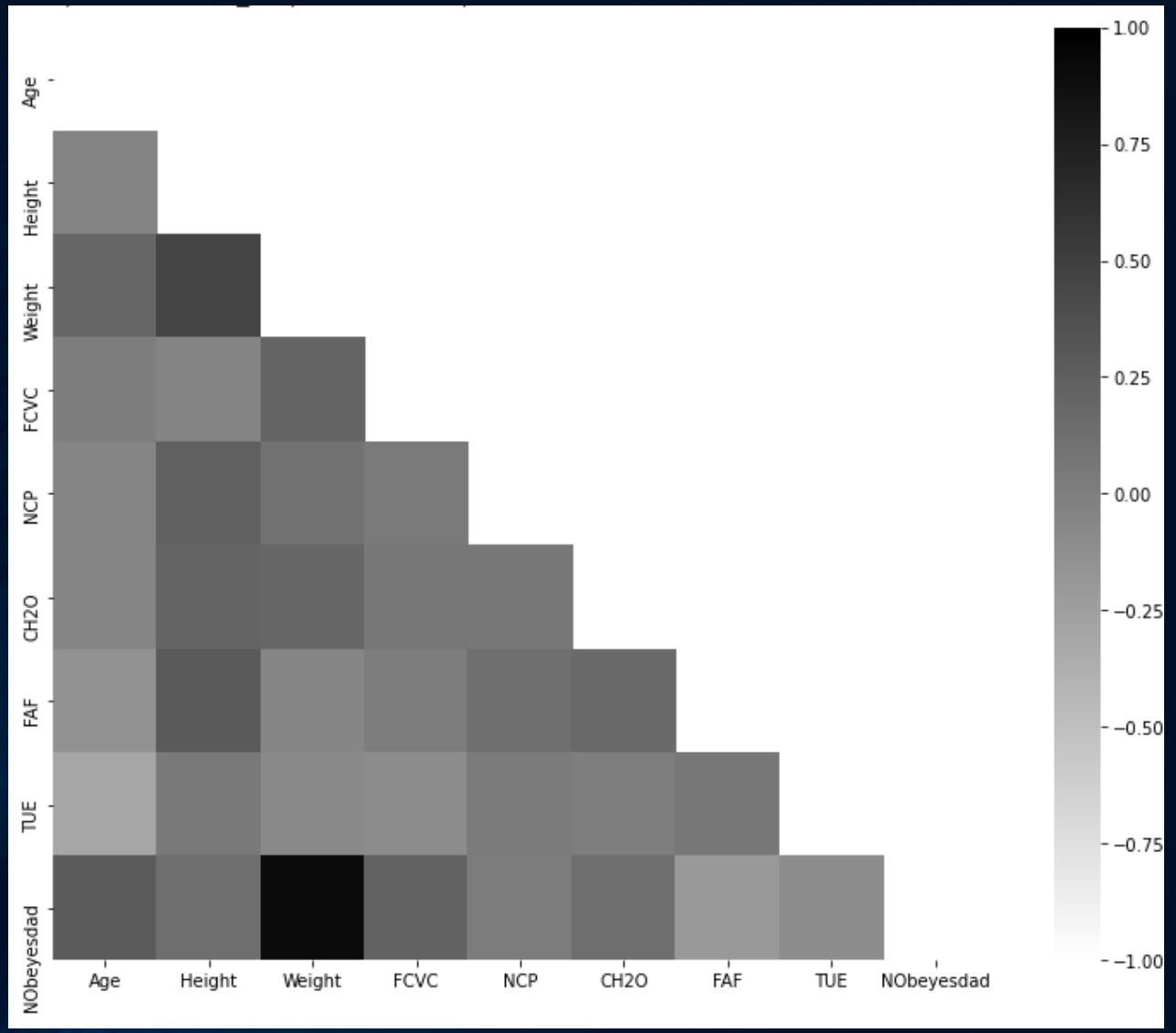
DISTRIBUTION DE FRÉQUENCE DU POIDS



Corrélation des variables

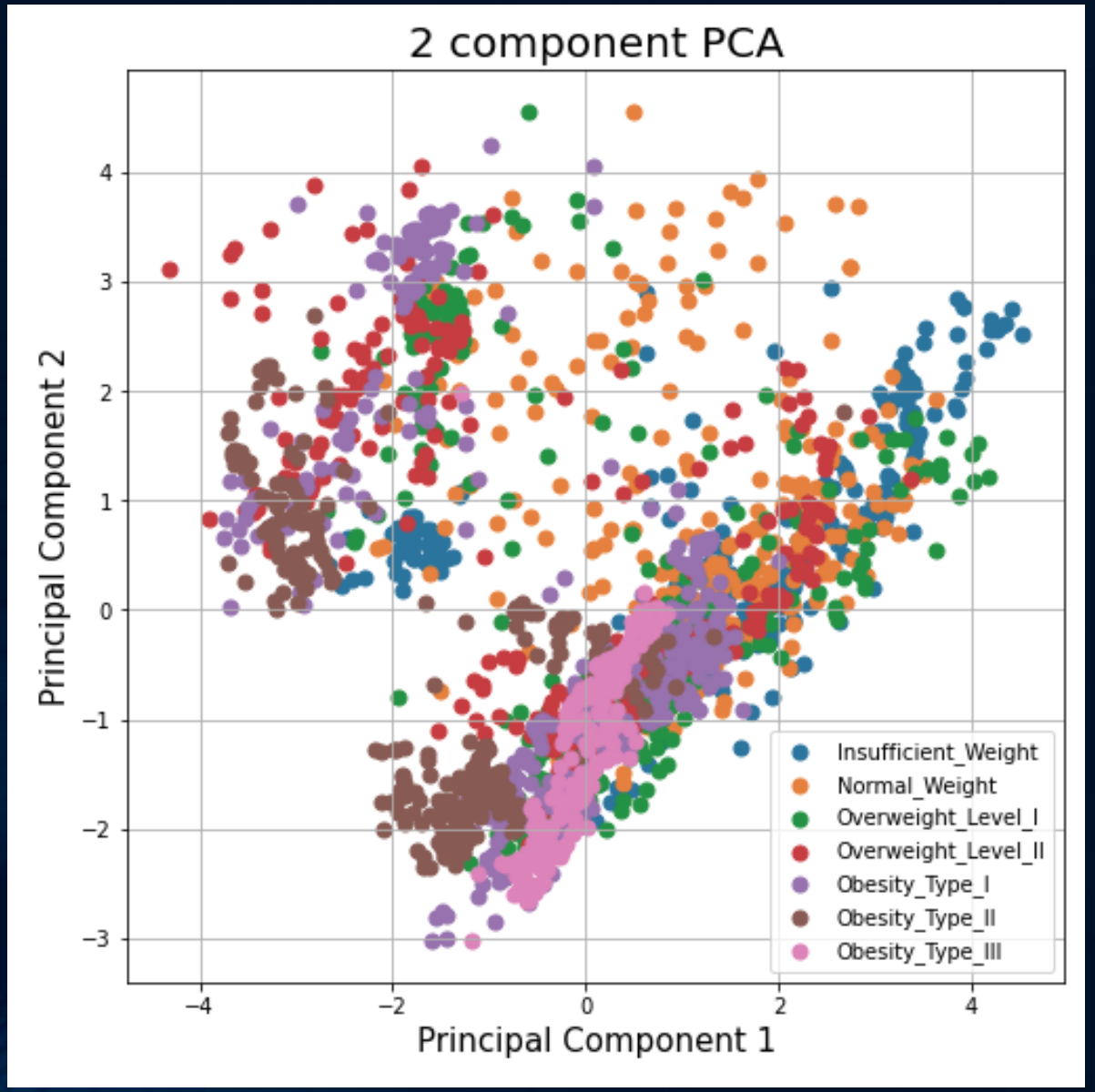
Le type d'obésité est très corrélé avec le poids, ce qui est logique.

Mais également avec l'âge et la consommation de légumes.



Principal Component Analysis

Les informations contenues dans une colonne sont le montant de la variance qu'elle contient. L'objectif principal des composants principaux est de représenter les informations de l'ensemble de données avec un minimum de colonnes possible.



II. Simplification du DataSet

CONVERSION DES DONNÉES DE STRING À FLOAT

```
my_data['CAEC'].replace(['Sometimes', 'Frequently', 'Always', 'no'],[1,2,3,0],inplace=True)
my_data['CALC'].replace(['Sometimes', 'Frequently', 'Always', 'no'],[1,2,3,0],inplace=True)
my_data['Gender'].replace(['Female', 'Male'],[0,1],inplace=True)
my_data['family_history_with_overweight'].replace(['yes', 'no'],[1,0],inplace=True)
my_data['FAVC'].replace(['yes', 'no'],[1,0],inplace=True)
my_data['SMOKE'].replace(['yes', 'no'],[1,0],inplace=True)
my_data['SCC'].replace(['yes', 'no'],[1,0],inplace=True)
```

AUCUN RETRAIT DE COLONNE N'A ÉTÉ EFFECTUÉ, CAR SEUL LE POIDS EST PRÉPONDÉRANT DEVANT LES AUTRES COLONNES

III. Création des modèles

PREMIER ESSAI AVEC UNE RANDOM FOREST

```
def applyModel(x_train, y_train,x_test,y_test, model):
    model.fit(x_train, y_train)
    y_pred  = model.predict(x_test)

    precision = precision_score(y_test, y_pred,average='micro')
    recall = recall_score(y_test, y_pred,average='weighted')
    f1score = f1_score(y_test, y_pred,average='macro')
    return ( model,precision, recall, f1score)
```

```
# Random Forest
RFcls = RandomForestClassifier(n_estimators = 10, oob_score = True,max_depth=None)
RFcls, RFprecision, RFrecall, RFf1score=applyModel(x_Train, y_Train,x_Test,y_Test, RFcls)
```

III. Création des modèles

DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier
DTclf=DecisionTreeClassifier()
DTclf, Summary_DT=PrintResults(DTclf, x_Train, y_Train,x_Test,y_Test, "DecisionTree")
```

SUPPORT VECTOR MACHINE

```
from sklearn import svm
SVMclf = svm.SVC()
SVMclf, Summary_SVM=PrintResults(SVMclf, x_Train, y_Train,x_Test,y_Test, "SVM")
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
LR, Summary_LR = PrintResults(LR, x_Train, y_Train,x_Test,y_Test, "LogisticRegression")
```

III. Création des modèles

STOCHASTIC GRADIENT DESCENT CLASSIFIER

```
from sklearn.linear_model import SGDClassifier
SGDclf = SGDClassifier(max_iter = 300, tol = None)
SGDclf, Summary_SGD = PrintResults(SGDclf, x_Train, y_Train,x_Test,y_Test, "SGD")
```

K-NEAREST NEIGHBORS

```
from sklearn.neighbors import KNeighborsClassifier

for n in range(2,12):
    KNNclf=KNeighborsClassifier(n_neighbors = n)
    KNNclf.fit(x_Train, y_Train)
    y_pred=KNNclf.predict(x_Test)

    precision = precision_score(y_Test, y_pred,average='micro')
    print("%.d Neighbors & Precision: %.2f %%"%(n,precision))
```

IV. Comparaison des modèles

LA RANDOM FOREST EST LE MODÈLE PLUS EFFICACE, NOUS ALLONS L'AFFINER EN MODIFIANT SES HYPER-PARAMÈTRES

	Model	Recall	F1-Score
Precision			
0.955083	RandomForest	0.955083	0.953769
0.947991	DecisionTree	0.947991	0.947501
0.895981	KNN	0.895981	0.885236
0.690307	LogisticRegression	0.690307	0.679375
0.612293	SGD	0.612293	0.572152
0.602837	SVM	0.602837	0.591347

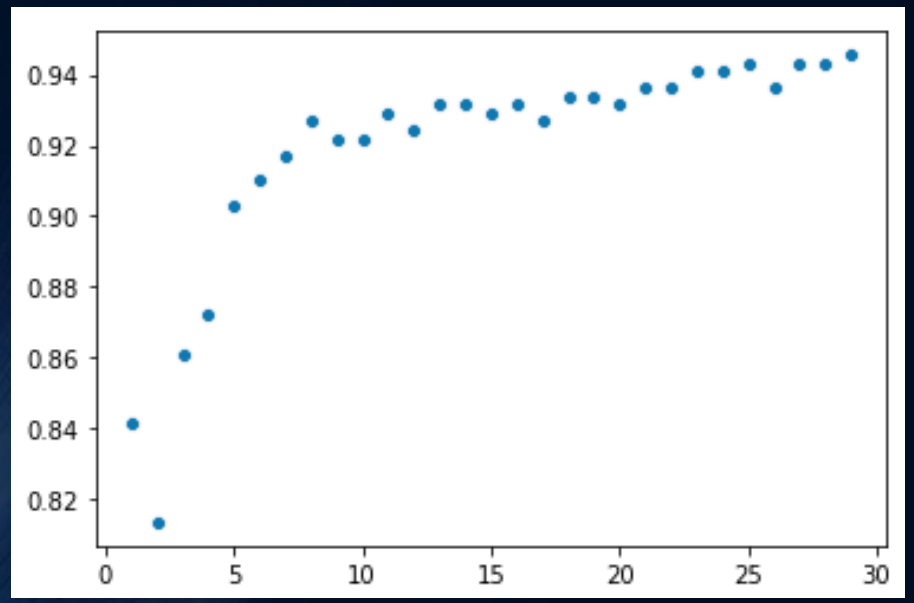
Rappel :

Le Recall est une mesure du nombre de cas positifs que le classificateur a correctement prédit, sur tous les cas positifs dans les données.

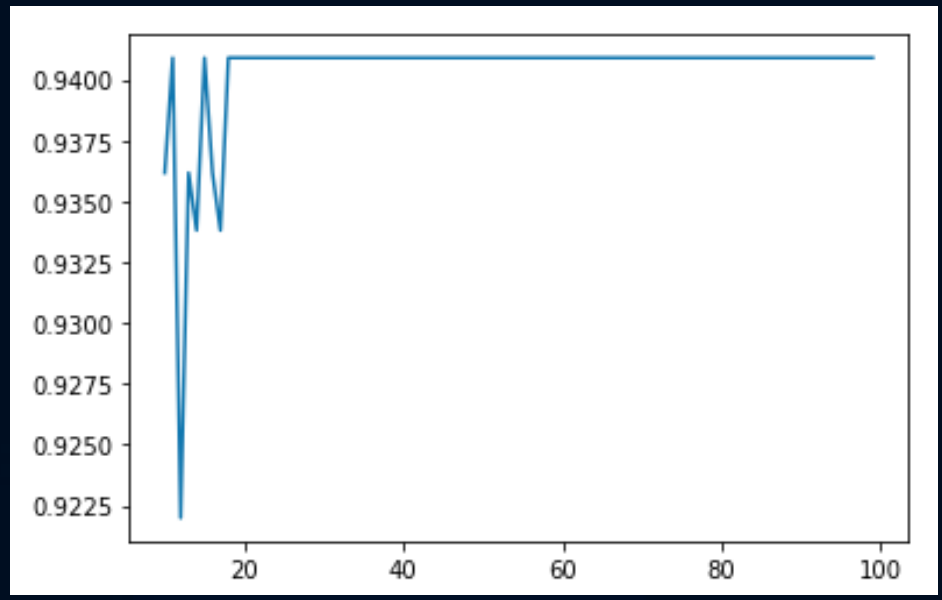
F1-Score est une mesure combinant à la fois précision et rappel. Il est généralement décrit comme la moyenne harmonique des deux.

V. Changement des hyper-paramètres

PRÉCISION EN FONCTION DU NOMBRE D'ESTIMATEURS



PRÉCISION EN FONCTION DE LA PROFONDEUR MAXIMALE



VI. Conclusion du modèle définitif

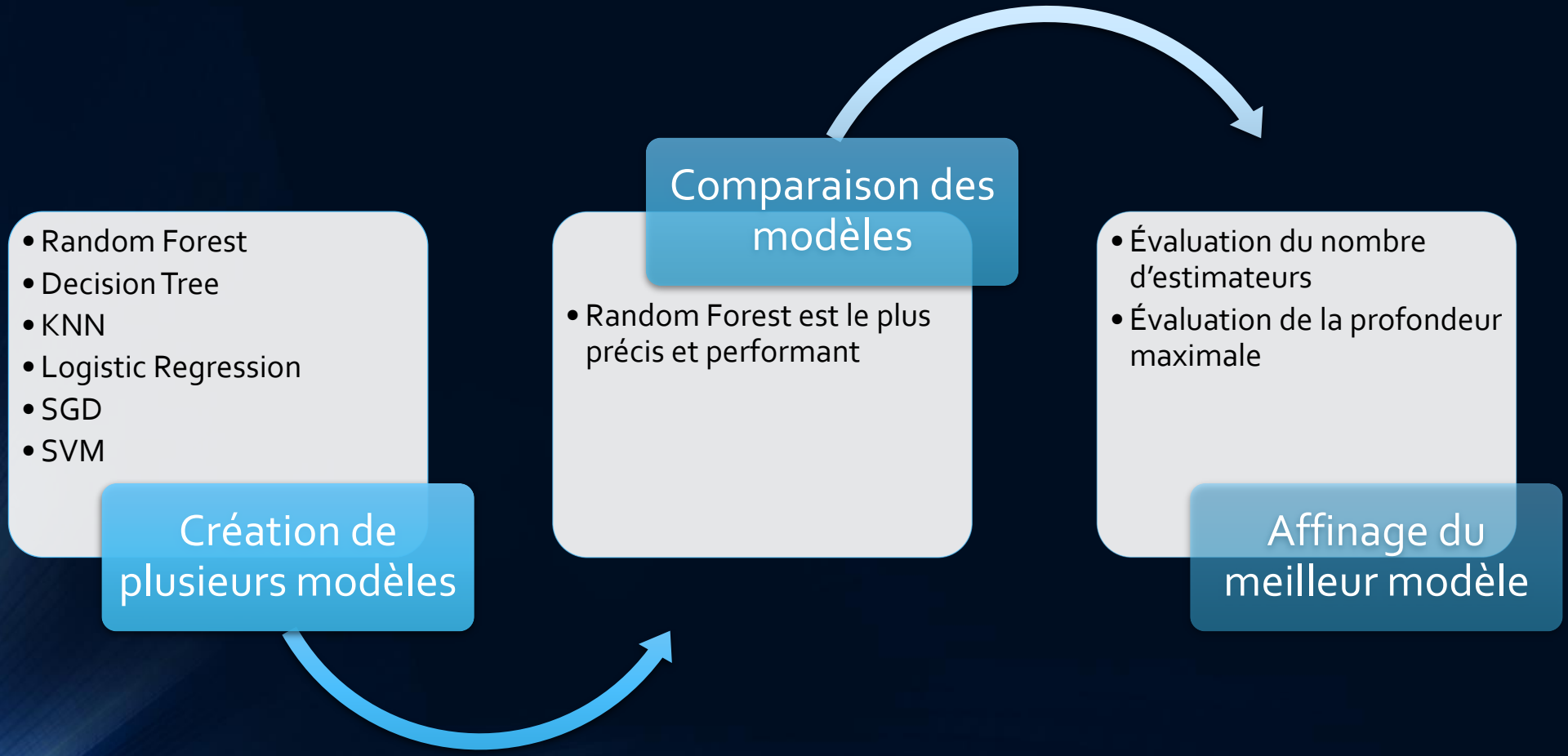
LA RANDOM FOREST EST LE MODÈLE LE PLUS PERFORMANT

```
RFclf = RandomForestClassifier(n_estimators = 24, oob_score = True,max_depth=40, random_state=2021)
RFclf, Summary_RF=PrintResults(RFclf, x_Train, y_Train,x_Test,y_Test, "RandomForest")
```

```
Precision: 0.9551 %
Recall: 0.9551 %
f1-score: 0.9538 %
```

ON RETIENDRA 24 ESTIMATEURS AINSI QU'UNE PROFONDEUR MAXIMALE DE 40 EN HYPER-PARAMÈTRES

Schéma global de notre démarche



BONUS : Test avec un réseau de neurones

ON OBTIENT UNE PRECISION MOYENNE DE 80,2%

```
def build_model():
    model = Sequential()
    model.add(Dense(16, input_dim = 20, activation = 'relu'))
    model.add(Dense(16, activation = 'tanh'))
    model.add(Dense(7, activation = 'softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer = 'rmsprop', metrics = ['accuracy'])
    return model
```

```
estimator = KerasClassifier(build_fn = build_model, epochs = 100, batch_size = 10, verbose = 0)
```

```
kfold = KFold(n_splits = 5, shuffle = True, random_state = 2021)
results = cross_val_score(estimator, x_Train, y_Train, cv = kfold)
```

```
print(results)
```

```
[0.84911245 0.84319526 0.76331359 0.79821956 0.75964391]
```

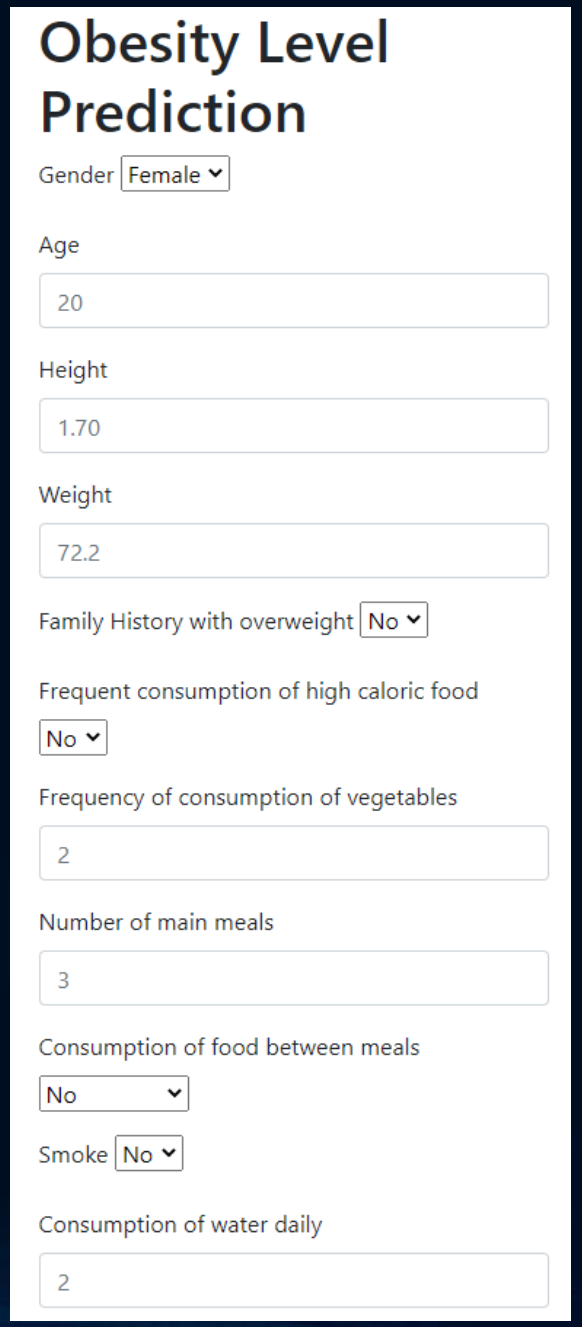
CE MODÈLE EST INTÉRESSANT, MAIS MOINS PERFORMANT QUE LA RANDOM FOREST

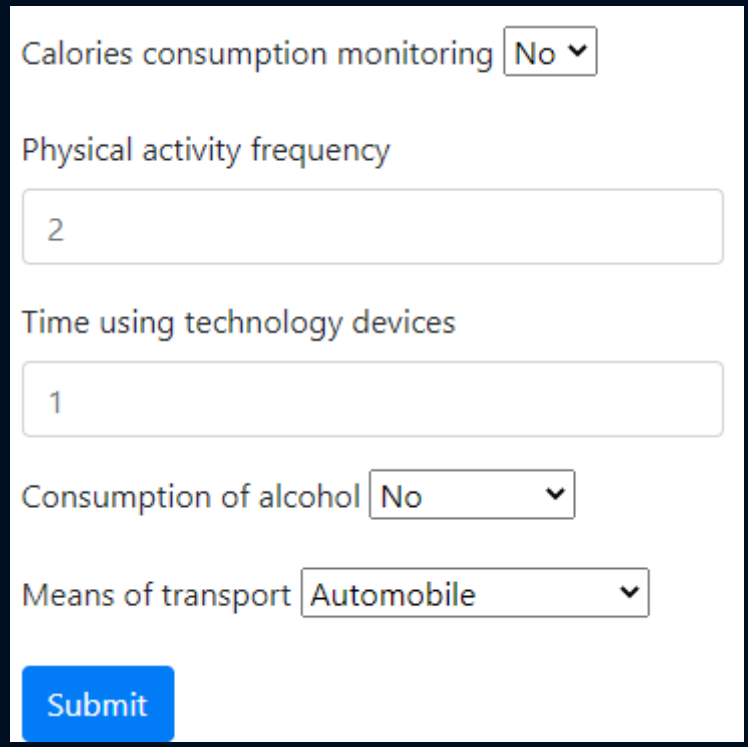
VII. API REST Django - header

ON RETROUVE EN HAUT, UN MENU DE NAVIGATION POUR CHOISIR DE PRÉDIRE LE NIVEAU D'OBÉSITÉ OU D'AFFICHER LA LISTE DE TOUTES LES PRÉDICTIONS QUI ONT DÉJÀ ÉTÉ FAITES.

VII. API REST Django – home page

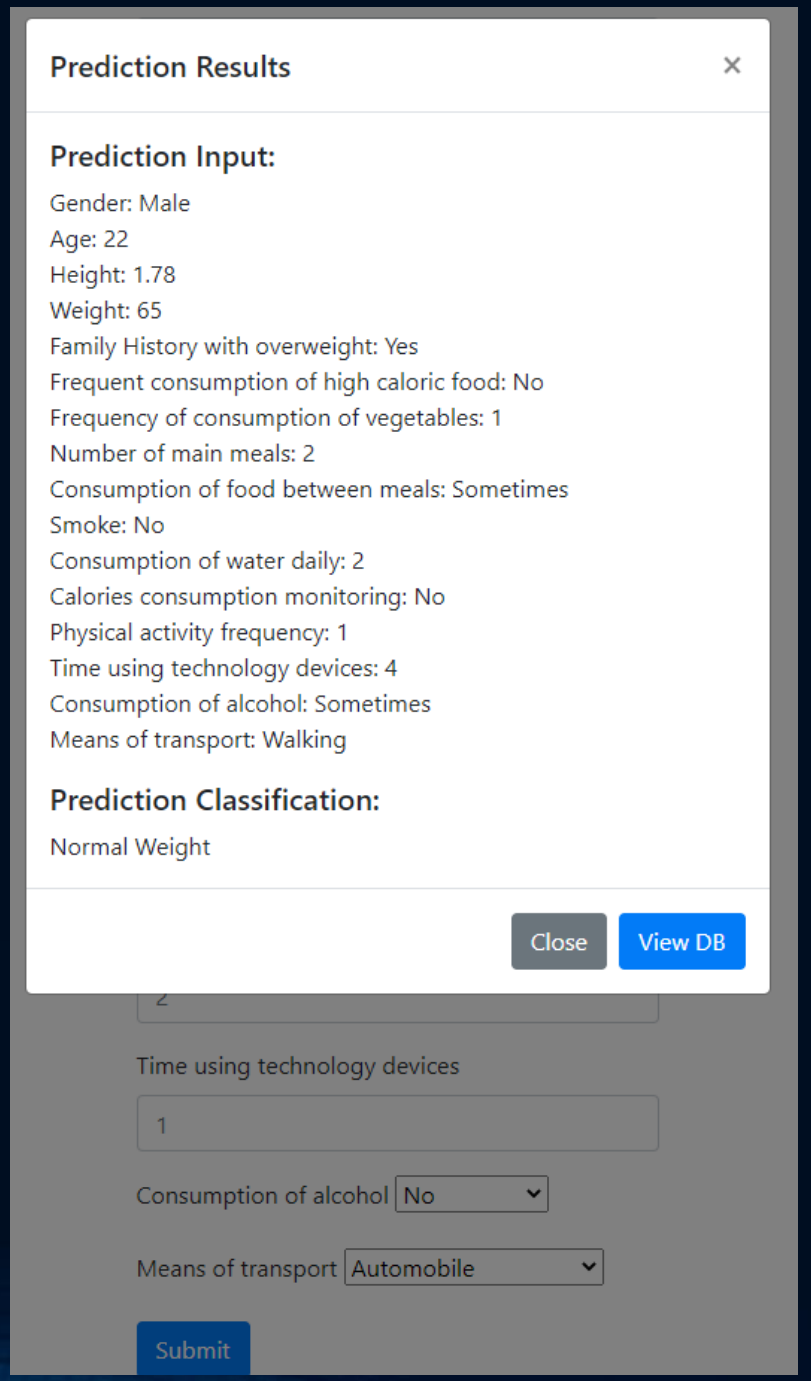
LA PAGE PRINCIPALE EST LA PAGE DE PRÉDICTION. L'UTILISATEUR DOIT ALORS REMPLIR TOUS LES CHAMPS PROPOSÉS AFIN D'OBTENIR UNE ÉVALUATION DE SON TYBE D'OBÉSITÉ.





VII. API REST Django – self prediction

L'ESSAI AVEC UN CAS DE FIGURE RÉEL S'AVÈRE PLUTÔT CORRECT.
LE MODÈLE ME PRÉDIT UN POIDS NORMAL ET MON IMC ME DONNE LE MÊME RÉSULTAT.



VII. API REST Django – Prediction Results

Prediction Results

ID	Gender	Age	Height	Weight	Family history with overweight	Frequent consumption of high caloric food	Frequency of consumption of vegetables	Number of main meals	Consumption of food between meals	Smoke	Consumption of water daily	Calories consumption monitoring	Physical activity frequency	Time using technology devices	Consumption of alcohol	Means of transport	Prediction of Obesity
1	Female	21.0	1.62	64.0	Yes	No	2.0	3.0	Sometimes	No	2.0	No	0.0	1.0	No	Public Transportation	Normal Weight
2	Female	24.0	1.5	60.0	Yes	No	2.0	3.0	Frequently	No	3.0	No	0.0	5.0	Always	Automobile	Normal Weight
3	Male	22.0	1.78	89.8	No	No	2.0	1.0	Sometimes	No	2.0	No	0.0	0.0	Sometimes	Public Transportation	Overweight Level II
4	Female	20.0	1.7	72.2	No	No	2.0	3.0	No	No	2.0	No	2.0	1.0	No	Automobile	Normal Weight
5	Female	21.0	1.7	84.0	No	No	2.0	3.0	No	No	2.0	No	2.0	1.0	No	Automobile	Normal Weight
6	Male	22.0	1.78	65.0	Yes	No	1.0	2.0	Sometimes	No	2.0	No	1.0	4.0	Sometimes	Walking	Normal Weight

L'ONGLET DB DONNE ACCÈS À TOUTE CETTE BASE DE DONNÉES, COMPOSÉE DES DIFFÉRENTES PRÉDICTIONS QUE L'APPLICATION A PU NOUS FAIRE.